DAQ Manual

Author: Jeff Landgraf (jml@bnl.gov)

Date: 6/6/2000
Revision: 1.0

Table of Contents

I. Overview of DAQ

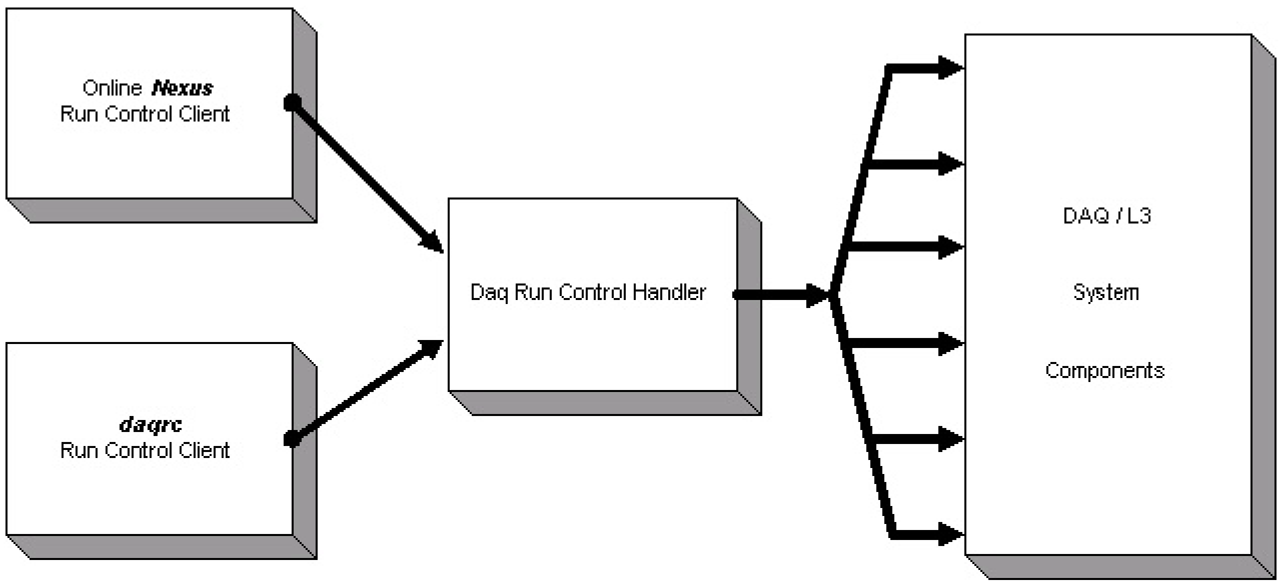
The STAR data acquisition system collects data from the trigger and detector systems and stores it into files on the RHIC computing facilities HPSS tape system. The chief requirement of the DAQ system is that it be capable of storing whatever data is produced by the detector systems at a rate approaching 20 MB/Second. In addition the DAQ system acts as the base for the STAR Level 3 Trigger, which is integrated into the real-time framework of the DAQ system.

A. Data Flow Paths

One way of describing DAQ is in terms of the flow of data through the system. In this document we will describe the flow of information through the DAQ system in terms of five related information paths. These include the path for control information, for data, for the L3 trigger, for monitoring, and for archiving in the online databases. Each function is related. Most of the components of DAQ have roles in several of the data paths, however each function also has specific, dedicated DAQ components and utilities.

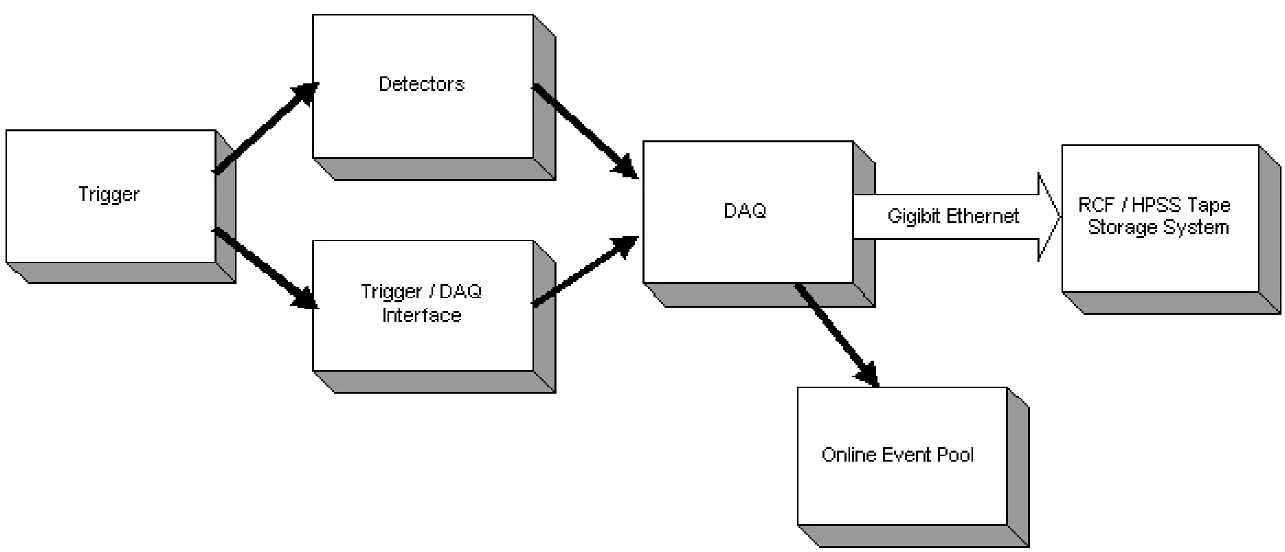
Control Path:

The first type of information is control information. This information starts and stops runs, and configures the DAQ system. Control information is produced by a run control client. There are currently two run control clients, the online ***Nexus*** run control and the DAQ command line utility ***daqrc.*** From the point of view of DAQ, the DAQ control line program ***daqrc*** is identical to the full online run control. In both cases, the run control program is a client of the DAQ run control handler. The run information is passed to the DAQ run control handler, and this program forwards the information to each of the components of DAQ:



Data Path

The second type of information that flows through DAQ is the experimental data.



Data flow into the DAQ system:

An event is passed into DAQ through two distinct channels. The first is through the data produced by the readout boards of the individual detectors and passed to the DAQ receiver boards through fiber optic cables. The second channel is through trigger/DAQ interface (TDI), through which the trigger sends information directly to the DAQ system. Although the signals are initiated in both cases by the trigger, each data path is independent. If DAQ does not receive valid signals from either the detectors or from the trigger it can not successfully build events. The DAQ system can run without the TDI link to trigger with reduced functionality.

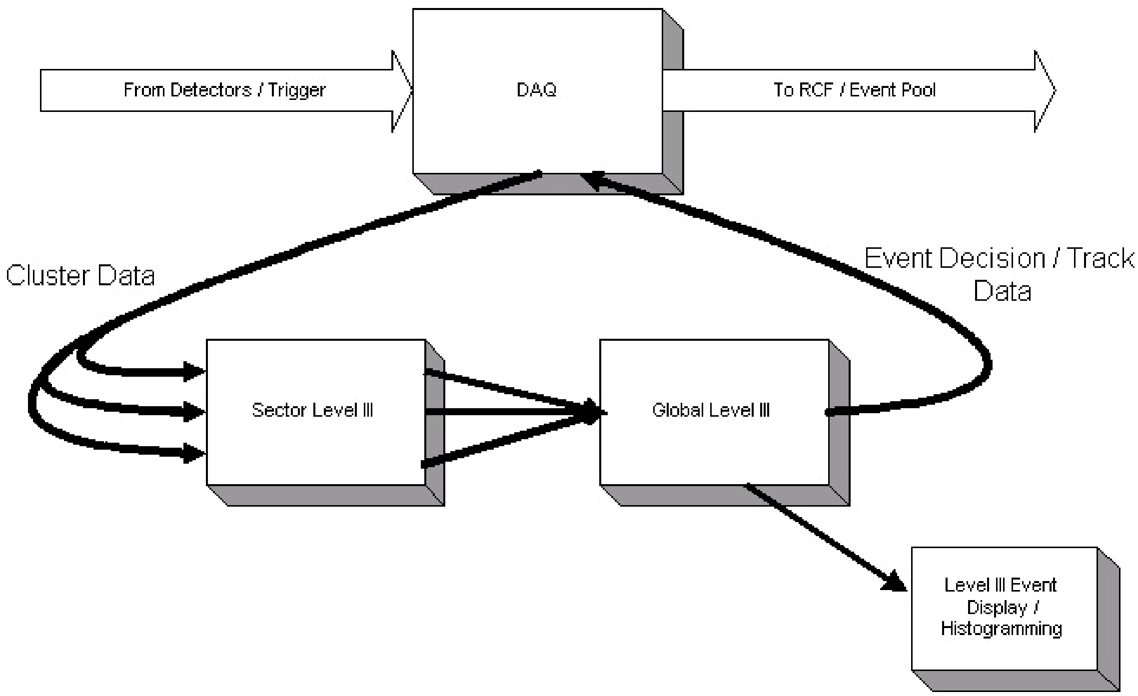
The internal components of DAQ and the Level 3 trigger communicate over a Myrinet Network. This network is used to send the messages that synchronize the operation of the DAQ hardware. It is also the means by which the data is distributed by the Detector Brokers to the Level 3 CPU's and to the Event Builder.

Data Flow out of the DAQ system:

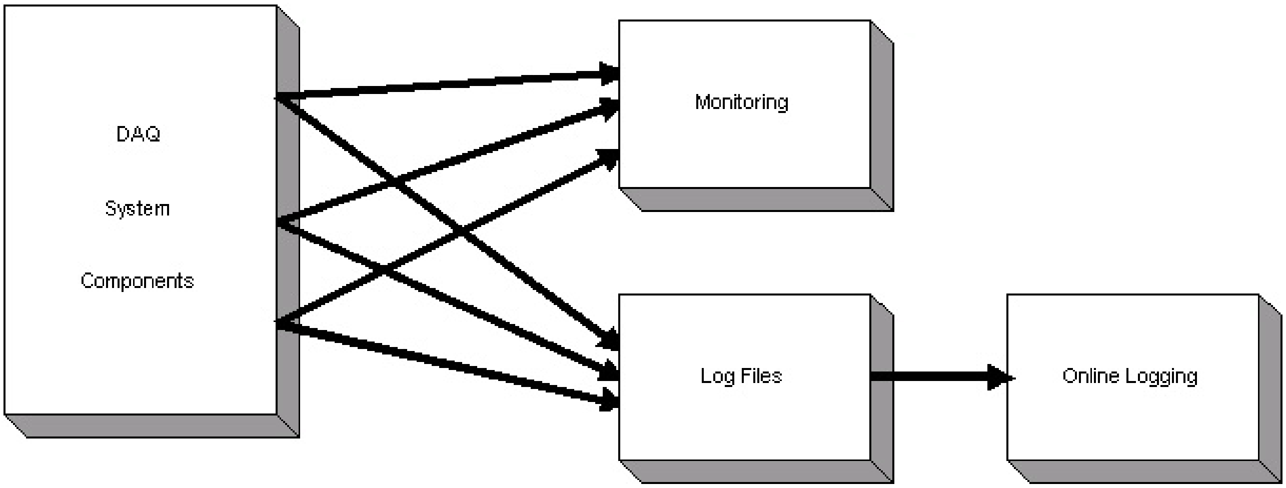
Once the event is in the DAQ system it must be stored. The normal running mode is for events to be stored directly into the RCF HPSS system. The online event pool also may request and receive events from DAQ.

Level Three Trigger Data Path:

The level three trigger runs as a part of the DAQ system. When an event arrives, the cluster data is passed to the L3 system. The L3 system tracks the data in real time and issues a decision whether to build the event or not.   This decision is then returned to DAQ. Although the L3 system runs as part of DAQ, DAQ can be configured to bypass L3 processing entirely, or too include the L3 processing without using it to reject events.



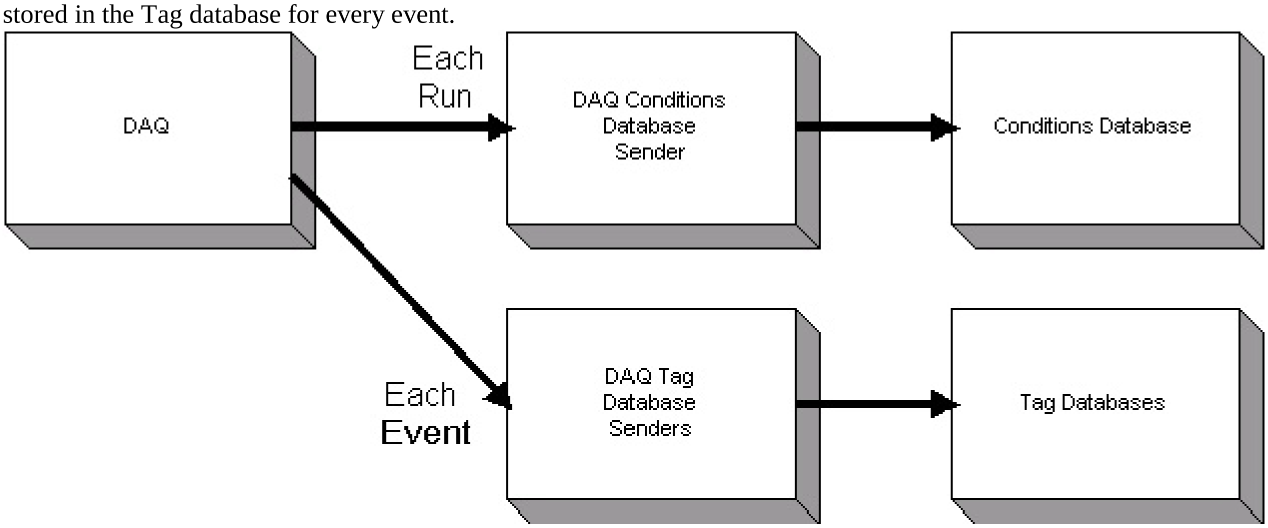
Monitoring / Logging Data Path:



Each of the DAQ components sends information to a set of monitoring programs. These programs display real-time information, such as the system state and the instantaneous data rates. In addition, all of the DAQ components send error messages and debugging information to a centralized log file, which can be forwarded to the online message logging service.

Database Data Paths:

DAQ saves its configuration data to the conditions database at the start of every run. In addition, a short data record is stored in the Tag database for every event.



B. DAQ Hardware Components

Receiver Boards (TPC, SVT, FTPC):

The DAQ Receiver Boards receive the data from the detectors through fiber optic cables. The boards each receive data from a single cable and process the data using three mezzanine boards. Each of the mezzanine boards contains 6 custom ASICs which process the raw data from the fiber and an I960 microprocessor which controls the ASIC, formats the raw data into the event structure, and produces clusters for the L3 trigger system. The Receiver Board is the most visible hardware used in the DAQ system. There are 6 receiver boards allocated to every TPC sector, and 12 receiver boards (2 sectors) in each VME crate. The LED's on the front of the receiver boards give information about the state of the system.

General Information LED's

+5V	Green LED	On if the +5V power is applied
+3.3V	Green LED	On if the +3.3V power is applied
RST	Red LED	On if the receiver board is being reset
VME Busy	Yellow LED	On if the VME Bus is busy. This LED flashes every event.

Mezzanine Information

MRES1	Red LED	On if mezzanine 1 is not configured
CBSY1	Yellow LED	Hardware busy for mezzanine 1
SBSY1	Yellow LED	Software busy for mezzanine 1
MRES2	Red LED	On if mezzanine 2 is not configured
CBSY2	Yellow LED	Hardware busy for mezzanine 2
SBSY2	Yellow LED	Software busy for mezzanine 2
MRES3	Red LED	On if mezzanine 3 is not configured
CBSY3	Yellow LED	Hardware busy for mezzanine 3
SBSY3	Yellow LED	Software busy for mezzanine 3

Link Information

WDT ERR	Red LED	On if watchdog timer fires
HDBAD	Red LED	On if data header was invalid
LNKERR	Red LED	On if error in data stream
BSYER	Red LED	On if an event comes while the busy is set
ABORT	Yellow LED	On if trigger issues an abort
LNK OK	Green LED	On if link is correctly synchronized
FIBOK	Green LED	On if good fiber connection

Busy information

ENA	Red LED	Busy is not enabled if LED is on
ON	Yellow LED	Busy Set

In general, Red lights mean that the receiver board is not able to accept data. Yellow means that a busy has been set, and green means that the system is ready.

Detector Brokers:

Each VME crate contains one CPU in addition to the Receiver Boards. This CPU is the Detector Broker. Its role is to manage the communication between the Mezzanine Boards and the DAQ system, to collect the data from the mezzanines, and to ship the data over the Myrinet network to the Sector L3 CPU Farm and to the Event Builder.

The Global Crate:

The Global Crate contains several additional CPU's, including the Event Builder, the Global Broker and the TDI. These CPU's are all involved in managing the DAQ real-time protocol, and in collecting the data into a single manageable source. It also contains the CPU that controls the RICH detector and the Busy Distribution Boards.

The Busy Distribution Boards:

The busy signal from each receiver board is accessible through a LEMO cable on the front panel. Each busy cable is routed to a Busy Distribution Board (BDB) mounted in the 6U portion of the VME crate. This board performs a logical OR on all of its input signals and passes the combined signal to a similar set of boards mounted in the Global Crate. Each of these boards contains 14 LED's. The top 13 LED's turn yellow if the corresponding input is busy. The 14th amber LED is the Live signal (the LED is lit if all of the board inputs are NOT busy.) There are four BDB boards in the Global Crate. The first two are for the TPC. The global TPC live light is the amber light at the bottom of the second board. The third board contains the busy for the SVT. The fourth board contains the busy for the FTPC. There is no global busy signal for the DAQ system.

The Buffer Box:

The Buffer Box is a Solaris Sun Computer introduced into the system primarily to provide a local buffer for data in the event that RCF is unavailable for short periods. The bufferbox is capable of buffering a little more than 2 hours of data (160Gb). It provides the Gigibit Ethernet link to the RCF HPSS system. It receives data from the Event Builder over the Myrinet and passes it to HPSS using a private Gigibit Ethernet link.

C. DAQ Software Components

The following software components run on the DAQ CPU's. They will be running automatically whenever the hardware they run on is switched on. They can be monitored with the *monClient* utility:

Detector Brokers (DET)

The Detector Broker is the component that handles the data from the detectors. It orchestrates all of the low level data-handling functions of the DAQ system. For the TPC there is one Detector Broker for every two TPC sectors. For detectors that produce less data, there may be only a single detector broker for each detector.

Trigger DAQ Interface (TDI)

The TDI runs on a CPU in the located in the Global Crate. This CPU connects to the Trigger with a simple point to point SCI network. The TDI receives tokens and the trigger data from the trigger system. It also returns the token to the trigger system after an event has been processed.

Token Manager (TM) / Global Broker (GB)

TM and GB keep track of the events that are in the DAQ system at any given time.

Event Builder (EVB)

The Event Builder receives data from all of the other components, builds the data into files, and passes it the Bufferbox software.

The following software components run on the L3 Linux farm.

Sector Level III (SL3)

The Sector Level Three processors perform tracking for data from a specific Detector Broker (Usually two sectors of the TPC).

Global Level III (GL3)

The Global Level Three merges the tracks from the SL3 processing and runs algorithms to produce an event decision. GL3 also has provisions for doing real-time analysis by producing histograms. It is also the source of data for the L3 Event Display. It is important to realize that the level 3 system can display and analyze data from the full set of events that are announced into the DAQ stream. This analysis can include events introduced into DAQ that do not show up in any other data stream.

The following components run on the Solaris machines daqman and bufferbox. These tasks will always be running. They can be monitored with the *monClientRC* utility.

Bufferbox Reader / Bufferbox Writer (BB)

The Bufferbox Reader reads the data from the event builder. The Bufferbox writer saves this data in the HPSS system at RCF. If RCF is unavailable for short periods, the Bufferbox can buffer the data and send it in an low bandwidth auxiliary stream in parallel with regular data taken once RCF availability returns.

Run Control Handler (RC)

The Run Control Handler is the server that the ***daqrc*** program and the online Run Control programs connect to. It distributes run control commands to all of the DAQ components, and ensures the consistency of the state of the DAQ system.

Database Senders (DB)

There are four programs that send data to the four databases. The Conditions Database Sender runs on daqman. There are three Tag Database Senders, one for each of the tables that make up the tag database. Each of these processes runs on the bufferbox.

D. Additional Documentation

For more details about the DAQ system you can study the following documents available from the web page http://daq.star.bnl.gov/~daq.

> **Raw Data Format –** Contains the specification for raw DAQ datafiles.
> **DAQ Protocol –** Contains the internal communication protocol for the DAQ system

II. DAQ Setup

A. Choosing and using a run control client

There are two clients available for starting and stopping runs. The first is the online group's ***Nexus*** run control program. The second is the DAQ command line client, ***daqrc.*** Both of these programs connect to the DAQ run control handler and pass it an identical configuration structure, so from the perspective of DAQ both clients are identical. However, the two programs set up the parameters with a very different interface. The two programs also have several complementary features.

For typical shift operation it is expected that you will use the online ***Nexus*** run control to start and stop runs. Its main feature is a GUI interface. Additionally, it integrates the DAQ and Trigger run control. It also automatically logs the parameters used to start the run in both the online run log database and the DAQ Conditions database.

On the other hand, the ***daqrc*** program does not run the trigger. It only logs the parameter set used into the DAQ Conditions database. It does have several debugging/monitoring features that are not available through ***Nexus.***

The run control handler accepts multiple attached client run control programs. Each of these programs will be assigned a client ID which is displayed at the ***daqrc*** command prompt, and which can be displayed by typing "query" at the command line. While any number of clients can be attached and execute queries about the configuration and state of the DAQ system, only one can issue commands. We call this client the "Big Cheese". A ***daqrc*** client can take over the system by issuing the command "cheese". It can give another run control (such as a ***Nexus*** session) command by issuing the command "cheese clientID".

B. Typical Run Configuration

Normally, while starting runs there will only be a small number of parameters which need to change from run to run. These are as follows:

Run Number – The run number should be automatically generated by the run control client. It is a seven-digit number with the format YDDDNNN. Y is the year measured from 1999 = 0. DDD is the day of the year of the run. NNN is a sequence number starting from 0 for the first run of the day and increasing by one for every run. Be careful if you are mixing the use of the ***Nexus*** and the ***daqrc*** programs. Each generate the run numbers internally, and so you will need to set the run number by hand when you switch programs to avoid overwriting run numbers.

Run Type – Every time the DAQ system is rebooted, the first run taken must be a pedestal run. The pedestal values must be available before the DAQ can perform any meaningful zero suppression. There are currently four run types:

- Pedestal Run – Calculate pedestal and RMS values
- Physics Run – Normal zero suppressed physics data
- Configuration Run – Special run for debugging the readout board & FEE connectivity
- Test Run – Expert only mode, with custom tailored DAQ behavior.

The typical pattern will be that a pedestal run will be taken, with 50 – 100 triggers. Then one or more physics runs will be taken. Special trigger types such as laser runs, or pulser runs are treated as physics data for DAQ.

Data Destination – You can choose to send the data to RCF or to not save the data at all. You can also choose to save the data locally on the bufferbox, or to save a local copy of the data. These options should not be chosen during normal data taking because of the large data sizes. If, for any reason, the HPSS system goes down, the data will automatically start going to local disk. In this case, there is space for approximately 2-3 hours of data on the bufferbox.

C. Additional Run Configuration

From run to run, it will sometimes be necessary to add / subtract detector systems from the run. Also, it is possible to remove certain DAQ system components and run with reduced functionality (For example we may often decide to run without the Level 3 subsystem). Furthermore, all of the DAQ parameters are organized by the system the parameter refers to. A few of the systems do not correspond to physical components that can be added or removed from a run, but instead serve only as a way to organize the parameters. For example, the magnetic field is given to DAQ as a parameter for the "Slow Controls" system. This only means that the source for the value for the parameter is associated with slow controls. It does not imply that the DAQ run control handler configures the magnet.

The following system types exist:

- DAQ – contains global parameters for DAQ
- Detector Systems - Contains parameters for each individual detector
  - TPC
  - SVT
  - FTPC
  - RICH
  - EMC
  - SMD
  - TOF
- Subdetector Systems – Contains parameters for each individual subdetector
  - TPC[1..24]
  - SVT[1..4]
  - FTPC[1..2]
  - RICH[1..2] – The RICH subdetector refers to the two RICH DAQ processors. #1 is in the DAQ room. #2 is on the platform. These processors can be configured independently.
- GL3 Systems – Contains general parameters for GL3
- GL3 Nodes[] – Contains parameters for each separate GL3 node
- SL3 Systems – Contains general parameters for SL3
- SL3 Nodes[] – Contains parameters for each separate SL3 node
- TDI System – For adding / subtracting TDI from the run (***daqrc*** only)
- SLOW – Parameters originating from slow controls
- CAL – Parameters originating from online analysis
- RUNDEFS – Default run type setup (***daqrc*** only)
- GLOBAL – Global parameters for the run control handler (***daqrc*** only)

The Detector/Subdetector systems can be added or removed from runs by the run control clients, as can the GL3 / SL3 systems and the TDI. The other systems are always present.

D. Parameters Listed by System Type

DAQ		
	run_type*	Specifies the type of the run
	run_number*	Specifies the current run number
	aux_bandwidth	A number containing the minimum events to wait between sending events to the event pool.
	event_number	obsolete
	dest_main	The internal protocol used to transfer data from the event builder to the bufferbox (Set to 6)
	dest_aux	The internal protocol used to transfer data from the event builder to the online event pool (Set to 5)
	event_suppression*	Save every –Nth event
	automate_es	(***daqrc*** only) Set the event_supression to -2000000 for pedestal runs and to –1 for physics runs
	use_tdi	Use the Trigger-DAQ interface
	use_gl3*	obsolete
	evb_log_level	Controls the error logging level on the EVB
	myri_log_level	Controls the logging level for the Solaris myrinet

		driver
	bb_log_level	Controls the logging level for the BB reader/writer components
	force_reset*	Force a reboot of the mezzanine processors at run start
	final_dest*	The final data destination.
	test_run	If this is variable is set, the data is routed to a temporary directory where it will be deleted.

Detectors		
	in_run*	Is the detector in the run?
	ped_mode*	Do pedestal subtraction?
	gain_mode*	Which type of gain (Linear, seesaw, logarithmic, corrected logarithmic)
	analysis*	Detector dependent. For TPC this determines whether to run the cluster finder
	l3_type	obsolete
	log_level	The error logging level for this detector
	n_sigma	A hit is defined as a value above PEDESTAL + N_SIGMA. This is currently ignored for the TPC.
	format*	Which data banks should be requested
	t0_format*	Which data banks should be requested for the run summary event (token 0)
	asic_seq_hi	(Only for detectors with the ASIC) Minimum Sequence – 1 of hits above the high threshold to be considered an asic cluster
	asic_seq_lo	Minimum Sequence – 1 of hits above the low threshold to be considered an asic cluster
	asic_thr_hi	The high threshold for determining asic clusters
	asic_thr_lo	The low threshold for determining asic clusters
	time_bin_lo	The first time bin used for cluster finding
	time_bin_hi	The last time bin used for cluster finding
	gain_adc_val	
	use_sl3*	Send data to SL3
Subdetectors		
	in_run	Is this subdetector in the run?
	rb_mask*	For detectors with receiver boards, which receiver boards are in the run
SLOW		
	bfield	The magnetic field
	t0_estimate	T0 as predicted by slow controls
CAL		
	t0	T0 as predicted from online analysis
	driftvelocity	Drift velocity as calculated from online analysis
	driftlength	Drift length as calculated from online analysis
	szerror	
	xyerror	
	xvertex	X vertex position as calculated from online analysis
	yvertex	Y vertex position as calculated from online analysis
	dxvertex	Error in X vertex position
	dyvertex	Error in Y vertex position
GL3		
	in_run	Is GL3 in the run?
	write_daq	Write tracks to DAQ stream?
	write_local	Write data to L3 private disk
	formats	What L3 data banks to build
	log_level	Log level for GL3

	accept_triggered	Send accept to DAQ for all triggered data
	accept_automatic	Automatically send accept to DAQ for all events
	algorithm[1..20]	Algorithm Id's (see structure below)
GL3_algorithm		(GL3 ALGORITHM[x])
	alg_id	L3 Algorithm Id
	GI1 .. GI5	Algorithm dependent integer parameters
	GF1 .. GF5	Algorithm dependent floating point parameters
SL3		
	in_run	In the run?
	algorithm	Which tracking algorithm?
	version	Tracking algorithm version?
	log_level	Logging level for SL3
	minhitspertrack	
	neta	
	nphi	
	maxchi2primary	
	rowinnermost	
	rowoutermost	
	rowstart	
	rowend	
	hitchi2cut	
	goodhitchi2	
	trackchi2cut	
	deta	
	dphi	
	ptminhelixfit	
	maxtime	
	tx0 .. tx9	expansion parameters
rundefs	**(by run type)**	(***daqrc*** only)
	tpc_ped_mode	default pedestal mode for this run type, TPC
	tpc_gain_mode	default gain mode for this run type, TPC
	tpc_analysis	default analysis for the run type, TPC
	svt_ped_mode	
	svt_gain_mode	As above, SVT
	svt_analysis	
	ftpc_ped_mode	
	ftpc_gain_mode	As above, FTPC
	ftpc_analysis	
	emc_ped_mode	
	emc_gain_mode	As above, EMC
	emc_analysis	
	smd_ped_mode	
	smd_gain_mode	As above, SMD
	smd_analysis	
	tof_ped_mode	
	tof_gain_mode	As above, TOF
	tof_analysis	
	rich_ped_mode	
	rich_gain_mode	As above, RICH
	rich_analysis	

* Parameters which are set automatically by the ***daqrc*** program as part of the run start/stop procedure, or by separate commands. These parameters should never be set using the **set** command in ***daqrc***.

III. DAQ User Environment

To use the DAQ system, log into daqman.star.bnl.gov from a x-terminal using the account: **operator**. The password is displayed near the DAQ terminals in the control room. At login, several screens should appear. These will include the DAQ monitoring system, and a screen containing the DAQ message log file. From this account, you can run any of the utilities described below.

IV. DAQ Monitoring / Logging Utilities

A. Logging

Most of the debugging and error messages produced by any component of DAQ go to a central file, /DAQ/log/daqlog.log. This file is automatically archived every day and it is stored for one week at which time the file is deleted. You can view this file in real time by right clicking on the desktop and choosing

```
Logging
```

Here is an example line from the file:

```
[001.001 09:20:10] (tEvbLoop): NOTICE: evb.C [line 463]: 25 sec and no message
```

The first two quantities inside the brackets refer to the last two digits of the ip address of the node that produced the message, and to the time the message was generated. The string inside the parenthesis refers to the process that generated the message. The next value, “NOTICE” is the importance of the message. Following the log level is the source file and line number that generated the message. Finally, there is a string describing the messages meaning.

There are 5 logging levels of increasing importance

- DEBUG– Information used to debug the system only.
- NOTICE – Informational messages, that require no action.
- WARNING – Unusual occurrences that might lead to trouble, but require no immediate action.
- ERROR – An error condition occurred. It is likely that at least one event was not recorded, or is corrupt.
- OPERATOR – A condition occurred that the operator needs to know about.
- CRITICAL – The system has a major problem, and will not recover without intervention.

The typical operator should ignore all messages with a level less than ERROR.

B. Online Logging Sender

The program

```
/DAQ/bin/RC/onlMsg [no_online] [no_print] [level=#]
```

reads the daqlog.log file and forwards messages to the online messaging service. This program should always be running in the background. If it is not, execute the above command string. The parameters are optional. The first, no_online, just prints the log file to the screen. The second, no_print, turns off local screen printing. The third, level, sets the minimum log level to send to online.

C. Monitoring

The DAQ Monitoring systems can be started from the operator account by right clicking the mouse and choosing

```
Monitoring
```

from the menu. This choice runs the programs */DAQ/bin/UTILS/monClient* and */DAQ/bin/UTILS/monClientRC* which provide real-time monitoring of the DAQ system.

monClient – This program monitors the main run-time DAQ processes. The Token Manager, Trigger-DAQ Interface, Global Broker, Event Builder, BB Reader and Writer, and all of the detectors and subdetectors are listed. If the system is present and operational, then it is displayed in normal text. If the system is not present or is currently not operational, then it is displayed in reversed text. The most important parameter displayed is the systems **state.** The state can have the following values:

NONE – The system is booting.
PRESENT – The system is booted, but has never been configured.
READY – The system has been configured and is ready to run.
RUNNING – The system is currently running.
PAUSED – The system is currently paused.

In addition, each of these states can be modified with a "-W" which means that the system is in the process of changing states.

The following values are displayed for each task:

Tkn Acc – The last token accepted into the system
Tkn Rls – The last token released from the system
Tkn Bad – The last event thrown out by the system
Evts In – Number of events currently being processed by the system
Evts Run – Number of events in this run so far
Evts All – Number of events since this system was started
Evts Bad – Number of events thrown out since the system was started
Busy% - An estimate of the % time the system was able to accept new events
Evts/sec – The number of events processed per second
EVB kB/sec – The data throughput to the main stream in kB/second
Aux kB/sec – The data throughput to the aux stream in kB/second. (Depends on system)
State – The system state
Node Id – The systems DAQ node id
Task Id – The systems task id

In addition, the BB_Reader task displays:

Run Number – The run number

And, the BB_Writer task displays:

local disk – If the data is being written locally, which disk.
RCF dir – If the data is being written to HPSS, which directory. (Base is ~starsink/raw/daq/)

monClientRC – This monitor client shows the state of the run control system and the database links. The run control summary contains the following fields:

State – The overall state of the DAQ system
Last StateCh – The date and time of the last state change
Run Type – The current run type
Run Number – The current run number
Data Dest – The current final data destination
Test Run – If this is set to "TESTING" the data may be deleted at any time
 # GL3 – The number of GL3 nodes in the run
 # SL3 – The number of SL3 nodes in the run

The display also lists which detectors are in the run, and what the values of the ASIC parameters are for the detectors that use the ASIC.

There are also four database senders listed. The monitor lists the following parameters:

Server – The name of the database server the data is being sent to
Port – The port of the destination database
Lst Time – The last time a record was sent
Lst Date – The last date a record was sent
Backlog – The number of records that are waiting to be sent

The main purpose of the database entries is to know if and when the database senders crash. If any of the database sender entries is displayed in reverse text, contact a DAQ expert.

D. Testing for Data File Corruption

You can check data files to ensure that they have a valid raw data structure using the utility:

```
/DAQ/bin/UTILS/daqscan Filename
```

This program navigates through all of the bank headers, and checks the consistency of the data format.

E. Retrieving Configurations from the DAQ Conditions Database

You can get the parameters used to configure an old run from the conditions database using the program:

```
/DAQ/bin/DB/conditionsReader RunNumber [EndRunNumber]
```

If you supply only one run number, the parameters for that run will be displayed. If you also supply the second parameter, the program will display the valid run numbers within that range.

F. Debugging DAQ Parameter Files

There are two utilities for debugging DAQ parameter files:

```
/DAQ/bin/RC/checkOnlConfig Filename
/DAQ/bin/RC/checkDaqConfig Filename
```

The online configuration file is a file the run control clients use to keep consistent. The DAQ configuration file is how the parameter values are communicated to the DAQ components.

V. *Using the DAQ Control Line*

The DAQ Control Line, ***daqrc***, allows you to exercise the full functionality of the DAQ system. To start a ***daqrc*** in a stand-alone session type:

```
/DAQ/bin/RC/daqrc
```

To start ***daqrc*** as a monitor tool for use with the online ***Nexus*** Run Control type:

```
/DAQ/bin/RC/daqrc online
```

You should see a command line prompt that reads:

```
DAQ RC(##)->
```

The number in the parenthesis is your connection ID. If there is an asterix next to the number, then this client is the "Big Cheese" which means that it is able to control the experiment. If not, you will need to type the command ***cheese*** to take control of the system if you want to do anything other than monitor or check the system configuration.

A. Clearing the State of DAQ

The most effective way to clear the state of the DAQ system is to issue the command **reboot.** This command resets all of the systems that are currently in the run. All DAQ vx-works nodes are rebooted when this command is issued. DAQ tasks that run in Unix processes are killed and restarted.

There are two cases in which the **reboot** command will not reset a node. First, **reboot** only reboots nodes that are in the run. If the system is not currently part of the run and you want to reset it, you must first add it to the run. Second, **reboot** only works if the node is alive and talking to the run control handler. It is possible for the vx-works nodes to hang, or to be switched off. In the unlikely event that this occurs, you must reset the node using slow controls, or by toggling the hardware switches.

B. Determining the State of DAQ

There are three commands used to determine the current state of the DAQ system.

query – This command gives you the overall state of the system.

query states – This command lists the separate state of each system that is in the run. Use it to determine which system is having trouble if there is a problem.

querytokens – This command gives the number of tokens currently in the system.

C. Setting up / Viewing the Configuration of DAQ

To view the systems in the run use the command **set.** The **set** command is the main command used to view and modify configurations. Used with no arguments, it lists the detector systems in the run.

Add / Subtract Commands:

You can add/subtract detector systems using the commands **add** or **subtract**. The argument of the command is the detector system name, or the subdetector name. For example:

```
DAQ RC(1*)-> add tpc
```

Adds the entire TPC to the run. The command:

```
DAQ RC(1*)-> subtract tpc[5]
```

Subtracts TPC hypersector #5 from the run.

Mask Command

When you type the command **set** you will see a hexadecimal number displayed to the right of each subdetector system. This number is the receiver board mask for that subdetector, and it determines which receiver boards are turned on. The convention for this number is that the least significant bit corresponds to the leftmost receiver board in the VME crate. The following table shows the correspondence between the receiver board mask, the receiver board number, and the readout board for the TPC.

TPC Hypersector N (N odd)

RB Mask Bit	Receiver Board # (From left)	Readout Board
0	1	Sector N, Readout 1
1	2	Sector N, Readout 2
2	3	Sector N, Readout 3
3	4	Sector N, Readout 4
4	5	Sector N, Readout 5
5	6	Sector N, Readout 6
6	7	Sector N+1, Readout 1
7	8	Sector N+1, Readout 2
8	9	Sector N+1, Readout 3
9	10	Sector N+1, Readout 4
10	11	Sector N+1, Readout 5
11	12	Sector N+1, Readout 6

There is a special command to set the receiver board mask:

```
DAQ RC(1*)-> mask tpc[1] 0xfff
DAQ RC(1*)-> mask tpc[1] + 1 2 3
DAQ RC(1*)-> mask tpc[1] – 5 6 7
```

In the first form, the receiver board mask for the specified subdetector is set to the hexadecimal value. The second and third forms allow the user to turn on and off boards by there receiver board number. In these forms, the **mask** command is incremental. Only the bits listed are affected. Any number of bits can be listed in a single command.

Set Command

The **set** command is the powerhouse command for viewing and updating run configurations. It can be used in several incarnations. To display the list of detector systems use the form:

```
DAQ RC(1*)-> set
DAQ RC(1*)-> set all
```

In the first form, the list of all detector systems in the run will be displayed. In the second form, the list of all detector systems available will be displayed.

To display the parameters associated with some detector system or subdetector system use the following forms:

```
DAQ RC(1*)-> set tpc
DAQ RC(1*)-> set tpc[5]
DAQ RC(1*)-> set rundefs pedestal
```

The first form shows all of the parameters associated with the TPC.   The second, shows the TPC parameters along with the parameters specific to hypersector 5.   The third form displays the parameters associated with the rundefs system for pedestal runs.

To modify the values for any parameter use the following forms:

```
DAQ RC(1*)-> set tpc asic_thr_hi 5
DAQ RC(1*)-> set tpc[5] rb_mask 4095
DAQ RC(1*)-> set rundefs pedestal tpc_ped_mode 0
```

The first form sets the value of the *asic_thr_hi* parameter to 5 for the tpc. The second sets the rb_mask for tpc hypersector 5 to 4095. The third sets the tpc_ped_mode parameter to 0 for the rundefs record for pedestal runs.

Run Type Definitions

The feature that makes the ***daqrc*** quick and easy to use is that it remembers the previous configuration values used. Normally, the vast majority of configuration parameters is constant for a reasonably large number of runs. However, there are a few parameters that change for each run type. These parameters are:

- The pedestal mode – Whether pedestal subtraction is performed
- The gain mode – Which type of gain is applied
- The analysis performed – For example, whether the cluster finder is enabled

The rundefs "detector system" is used store these parameters for each detector by run type. When a run is started with a specific run type, the parameters from the rundefs records with the appropriate run type are copied onto the configuration file for each detector. The result is that the user of the system need only specify the run type when starting a run.

Starting a Run

To start a run you must first ensure that the proper detectors have been added to the system, that the system configuration is defined properly, and that the system is in either the "PRESENT" or "READY" state. Then, type:

```
DAQ RC(1*)-> startrun
```

The ***daqrc*** program will display the next run number. Normally, you should accept the default by pressing return. If this is the first run started using ***daqrc*** after runs started with the online ***Nexus*** program, you will have to choose a run larger than the last run to avoid overwriting datafiles.

The ***daqrc*** program will now ask you to enter the run type. Enter the appropriate run type. If this is the first run since resetting one or more nodes, or if you have changed the receiver board mask, you must choose a pedestal run.

The ***daqrc*** program will now ask for the final data destination. Enter the number for "Don't save" or for "Save to RCF".

Finally, the ***daqrc*** program will ask if you want to add a comment. If so, type "y" and enter the comment. To finish the comment enter "." on a line of its own.

The DAQ system will now start. When the command prompt returns, you may begin issuing triggers.

Stopping the Run

To stop the run use the command **stoprun.**

Exiting the DAQRC program

You can exit the ***daqrc*** program by using the command **quit.**

VI. Troubleshooting

A. Restarting DAQ Systems

The DAQ systems usually will not require any startup action, as they run all the time. If there is a serious error, it is possible that some system may go down. If so, follow these procedures to restart.

Restarting the Run Control Handler

If you are unable to start ***daqrc*** you probably need to restart the DAQ run control handler. If ***daqrc*** is operating properly, the DAQ run control handler does not need to be restarted. To restart the handler, follow these steps:

1. Log in to *daqman.*
2. Make sure that the handler is not already running with the command "ps –ef |grep handler"
3. If there the handler is already running kill it with "kill –9 *pid*" where *pid* is the handlers process id.
4. Type: /DAQ/bin/RC/starthandler.sh
5. The Run Control Handler should now be running. Try to connect to it using ***daqrc.***

Restarting the Bufferbox Reader/Writer

If the BB Reader and BB writer tasks show up in reversed text on the DAQ monitor for longer than a few seconds the Bufferbox code probably needs to be restarted

1. Log in to *bufferbox*
2. Type ps –ef |grep eth
3. Type ps –ef |grep bb
4. Type ps –ef |grep myri
5. Type ps –ef |grep rcf
6. Check for the following processes: ethComReader, ethComWriter, bbManager, rcfWriter, bbReader, bbReaderMonClient, bbWriterMonClient, bbMother, bbRepeat.sh.
7. If any of these processes are running kill them, starting with the bbRepeat.sh process.
8. Execute the script bbRepeat.sh &

Restarting Database Senders

If the any Database Sender tasks show up with reversed text on the DAQ monitor, then they need to be restarted. You first need to log on to the appropriate computer.

FILE_TAG – *bufferbox*
FILE_UPDATE - *bufferbox*
EVENT_TAG – *bufferbox*
RUN_TAG – *bufferbox*
CONDITIONS – *daqman*

Then, execute one of the command lines, as appropriate to the database that needs to be restarted

```
/DAQ/bin/DB/tagfilesSender &
/DAQ/bin/DB/tagfilesUpdateSender &
/DAQ/bin/DB/eventtagSender &
/DAQ/bin/DB/runtagSender &
/DAQ/bin/DB/conditionsSender &
```